2019 Computer Vision Project Report

Mascherin Alessandro, Scarcia Rossella

1. INTRODUZIONE

Lo scopo del progetto è la realizzazione di un'applicazione desktop che integri delle funzionalità di eye-tracking ottenute tramite OpenCV con l'interazione tramite un assistente vocale quale Google Assistant o IBW Watson. L'assistente vocale scelto è stato Google Assistant. Utilizzando OpenCV l'applicazione riesce a:

- Individuare la presenza di una persona all'interno della scena inquadrata da una webcam connessa al computer
- Effettuare il tracking della posizione del volto della persona all'interno della scena inquadrata
- Riconoscere la presenza e la posizione degli occhi, per individuare quando la persona stia effettivamente guardando nella direzione della camera.

L'interazione vocale da parte dell'assistente scelto avviene solo quando l'utente sta guardando nella direzione della camera per un certo periodo di tempo.

2. IMAGE PROCESSING

Per migliorare il riconoscimento delle feature del volto e degli occhi da parte degli algoritmi di ML utilizzati, sono state applicate delle tecniche per attenuare il rumore presente nelle immagini o difetti di acquisizione. Ogni frame registrato dalla camera viene convertito in scala di grigio e ne viene equalizzato l'istogramma. Successivamente viene applicato un ulteriore filtro bilaterale, che diminuisce il rumore mantenendo i bordi abbastanza nitidi. Sono state provate altre tecniche, quali l'utilizzo di un filtro Gaussian Blur per migliorare la qualità complessiva dell'immagine o la non applicazione dell'equalizzazione. Tuttavia questi filtri hanno prodotto risultati inferiori alle attese rispetto a quelli adottati.

Il frammento di codice presenta le funzioni OpenCV adottate nella configurazione finale del progetto.

```
def preprocess_frame(self, frame):
    grayImg = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    equalizedImg = cv2.equalizeHist(grayImg)
    # bilateral filter that remove noise but keeps edges sharp
    bilateral = cv2.bilateralFilter(equalizedImg, 5, 75, 75,
                                    cv2.BORDER_DEFAULT)
    return bilateral
```

I frame così elaborati vengono utilizzato per effettuare il riconoscimento facciale. Per questa operazione è stato scelto il classificatore di Haar sfruttando, fra i vari trained set più diffusi, il modello *"haarcascade_frontalface_alt2"*. Tale classificatore individua all'interno della scena porzioni rettangolari dello spazio che presentano delle Haar feature contenute nel training set utilizzato. Si è optato per considerare solo il rettangolo di dimensioni maggiori, che dovrebbe rappresentare il volto più vicino alla camera. Questo per evitare riconoscimenti spuri nello sfondo o per evitare ambiguità in presenza di due o più persone.

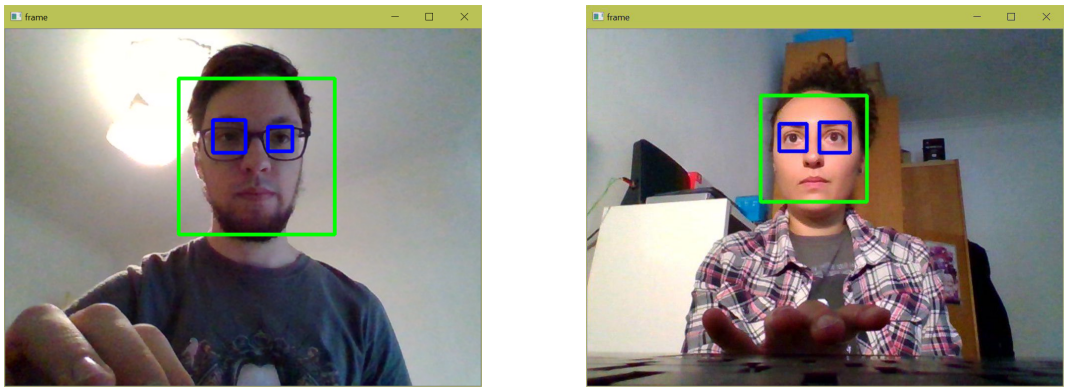
Per determinare la posizioni degli occhi nella scena, il rettangolo selezionato tramite il primo classificatore viene utilizzato come ROI per effettuare un'ulteriore operazione di riconoscimento. Anche questa seconda operazione viene realizzata utilizzando un classificatore di Haar, con un diverso modello: *"haarcascade_eye_tree_eyeglasses"*. Il risultato del classificatore viene considerato valido solo in presenza di entrambi gli occhi.

Le parti rilevanti di codice sono le seguenti:

```
def haar_eyes(self, img, equalizedImg):
    eyeClassifier = cv2.CascadeClassifier(eyeClassifierPath)
    coords = eyeClassifier.detectMultiScale(equalizedImg)
    # verifica della presenza degli occhi
    if coords is None or len(coords) != 2:
        return None
    else:
        return coords

def haar_face(self, img, equalizedImg):
    faceClassifier = cv2.CascadeClassifier(faceClassifierPath)
    rectangles = faceClassifier.detectMultiScale(equalizedImg)

    # Selecting only the biggest rectangle found with the face
    classifier
    selected_rect = (0, 0, 0, 0)

    if len(rectangles) > 1:
        for i in rectangles:
            if i[3] > selected_rect[3]:
                selected_rect = i
        selected_rect = np.array([i], dtype=np.int32)
    elif len(rectangles) == 1:
```

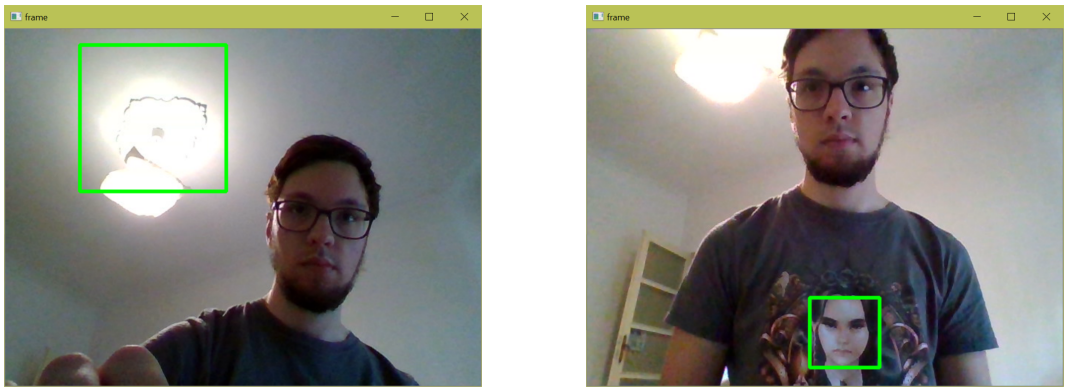
```
23          selected_rect = rectangles
24          return frame, eqFrame
25      else:
26          return None, None
27
```

Vengono ora riportati alcuni esempi di tracking corretto ed errato effettuato dall'applicazione. Il riquadro verde indica l'area riconosciuta come volto dal primo classificatore, mentre i riquadri blu rappresentano gli occhi.



(a) (b)

Figura 1: Esempi di tracking effettuato correttamente, in condizioni di controluce e con occhiali (immagine a) e con una buona illuminazione (immagine b)



(a) (b)

Figura 2: Esempi di problematiche nel riconoscimento del volto o degli occhi

3. IMPLEMENTAZIONE

3.1 Assistenti Vocali

Sono stati valutati due assistenti vocali da integrare nel progetto: IBM Watson e Google Assistant. Per lo scopo del presente progetto sono risultati essenzialmente equivalenti, pertanto si è optato per l'adozione di Google Assistant per la maggior disponibilità di documentazione presente in rete. Per utilizzare la Google Assistant API è necessario creare un nuovo progetto su Google Cloud Platform. Da qui è possibile selezionare la lingua che verrà poi usata dall'assistente vocale e collegare un progetto DialogFlow in cui vengono definite le interazioni dell'assistente.

DialogFlow fornisce di default la possibilità di avere una funzionalità di "small talk" con l'assistente, che può essere sovra-scritta o integrata con azioni personalizzate definite dal programmatore. Queste funzionalità nel contesto di DialogFlow sono definite *Intenti*. A scopo dimostrativo è stato aggiunto un ulteriore intento il cui scopo è fornire all'utente alcune informazioni sulla Computer Vision. All'intento sono state associate delle "training phrases", ovvero domande di esempio che il sistema utilizza per riconoscere l'intento e rispondere correttamente all'utente. È possibile definire anche dialoghi più complessi, o risposte dinamiche basate sullo stato del sistema. Queste feature non sono però state sviluppate, in quanto non obiettivo primario del progetto.

3.2 Google Assistant API

L'API di Google Assistant prevede un'interazione tramite stringhe testuali. Per avere uno "smart assistant" basato su comandi vocali, bisogna integrare nel progetto delle funzionalità di *speech-to-text* e viceversa, entrambe presenti in ulteriori API fornite da Google.

La Speech Recognition API è stata utilizzata per registrare la voce dell'utente e convertirla in una stringa. La lingua configurata per questa API è stata l'inglese, in quanto molte feature della Google Assistant API non sono disponibili in italiano. Le stringhe di testo ottenute vengono inviate a DialogFlow per ottenere una risposta da parte dall'assistente, anch'esse sotto forma di stringhe testuali.

Per la sintesi vocale è stata implementata gTTS (*Google Text-to-Speech*), una libreria Python che interfacciandosi con Google Translate produce un file .mp3 a partire da una stringa di input. Il file audio viene riprodotto con il modulo 'mixer' della libreria PyGame.

3.3 Gestione Thread

Le operazioni descritte nel precedente paragrafo sono sincrone e bloccanti. Nel momento in cui viene fatta partire una registrazione con la speech recognition API il thread corrente si blocca fintanto che non rileva un input vocale da parte dell'utente. Ciò impedirebbe una contemporanea acquisizione video tramite webcam necessaria per le operazioni di face recognition già descritte. È stato quindi adottato un semplice sistema di multi-threading, con un thread dedicato alle operazioni di Computer Vision e riconoscimento immagini e un secondo impegnato a gestire Google Assistant e conversioni di testo e voce.

3.4 Integrazione OpenCV e Google Assistant

Per permettere la comunicazione fra il sistema di classificazione delle immagini basato su OpenCV con le interazioni sviluppate per Google Assistant è stata introdotta una variabile globale condivisa fra i due thread. Quando viene riconosciuto un volto e due occhi all'interno dei frame analizzati dai classificatori, la variabile viene impostata affinché la comunicazione con l'assistenza sia abilitata. In particolare, il microfono e la collegata libreria *speech-to-text* rimarranno sempre attive, mentre le richieste inviate a DialogFlow avverranno solamente in presenza del riconoscimento facciale. Nel caso in cui dell'audio venga rilevato in assenza della presenza di un utente, l'assistente suggerirà di guardare la webcam per interagire con esso.

Il codice illustra le operazioni chiave che avvengono nel thread che gestisce DialogFlow:

```
# variabile globale condivisa fra i thread
global is_looking
while True:
    #libreria speech-to-text di google
    text_to_be_analyzed = self.recordAudio()
    if is_looking:
        text_input = dialogflow.types.TextInput(
                    text_to_be_analyzed,
                    self.DIALOGFLOW_LANGUAGE_CODE
                    )

        query_input = dialogflow.types.QueryInput(text_input)
        response = self.session_client.detect_intent(
                    self.session,
                    query_input)

        response = response.query_result.fulfillment_text
        self.speak(response) #gTTS
```

Poiché il riconoscimento da parte dei classificatori non sempre è preciso, questo può creare difficoltà nell'interazione con l'assistente vocale. In particolare, è stato osservato che a volte vengono riconosciuti dei volti "falsi", ovvero classificazioni spurie dovute a rumore o scarsa qualità della webcam. Anche in condizioni ottimali, qualche frame potrebbe essere scartato dal sistema in quanto il volto potrebbe non essere stato rilevato. Per ovviare a queste problematiche, si è pensato di introdurre una lista degli ultimi N frame analizzati dai classificatori. Se in questi N frame, sono stati riconosciuti almeno M volti, con al massimo un volto per frame, allora il sistema risulta stabile e viene avviata l'interazione con l'assistente.
In seguito a varie prove, con una camera a 30 fps, impostando N pari a 20 e M pari a N/2, si è ottenuto un buon livello di stabilità e precisione nel riconoscimento. L'interazione non sarà immediato, ma anche in condizioni di scarsa illuminazione, si riesce ad ottenere una comunicazione accettabile.

4. CONCLUSIONI

Nel complesso l'integrazione risponde agli obiettivi definiti in fase di progettazione. Rimangono alcune limitazioni dovute alla scarsa qualità delle web-cam e dei microfoni utilizzati, che sono risultati la causa principale di riconoscimenti errati, in particolare in condizioni di scarsa luminosità o contro-luce.

Tuttavia, cercando di adattare gli algoritmi applicati in fase di preprocessing a queste condizione limite, il tracking nel caso generale risulta più instabile. Si è scelto quindi un compromesso nella selezione di algoritmi di miglioramento immagine e il numero di frame considerati "validi" per abilitare la comunicazione.

L'applicazione è stata sviluppata con un'interfaccia grafica per mostrare il tracking facciale, che può però essere disabilita nel caso si desideri integrarla in sistemi di assistenti vocali più complessi.

Come possibili migliorie al progetto, si potrebbe integrare agli esistenti assistenti vocali presenti sui dispositivi mobile o in ambienti di domotica come Google Home o Amazon Alexa.